**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по практической работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Киреев К.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

## Цель работы.

Ознакомиться с основными характеристиками и особенностями такой структуры данных, как бинарное дерево, изучить особенности ее реализации на языке программирования C++. Разработать программу, использующую бинарное дерево для обработки формулы.

## Задание.

Формулу можно представить в виде бинарного дерева («дерева-формулы») согласно следующим правилам:

* формула из одного терминала представляется деревом из одной вершины с этим терминалом;
* формула вида (*f*1 *s* *f*2) представляется деревом, в котором корень − это знак *s*, а левое и правое поддеревья − соответствующие представления формул *f*1 и *f*2.

Вид формулы в постфиксной форме:

< формула > ::= < терминал > | ( < формула > < формула > < знак > )

< знак > ::= + | − | \*

< терминал > ::= 0 | 1 | ... | 9 | *a* | *b* | ... | *z*

Требуется:

* с помощью построения дерева-формулы t преобразовать заданную

формулу f из префиксной формы (перечисление узлов t в порядке КЛП) в инфиксную;

* упростить дерево-формулу t, заменяя в нем все поддеревья,

соответствующие формулам (f + 0), (0 + f), (f - 0), (f \* 1), (1 \* f), на

поддеревья, соответствующие формуле f, а поддеревья, соответствующие формулам (f \* 0) и (0 \* f), - на узел с 0.

## Основные теоретические положения.

Арифметическое выражение с бинарными операциями можно представить в виде бинарного дерева. Пусть, например, дано арифметическое выражение в инфиксной записи: (*a* + *b*) \* *c* − *d* / (*e* + *f* \* *g*). На рис. 1 представлено соответствующее ему бинарное дерево.

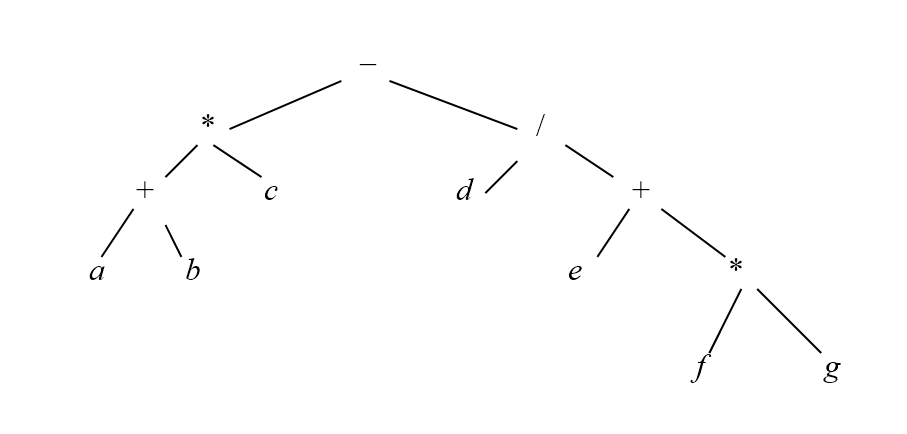


Рисунок 1 - Бинарное дерево, представляющее выражение

Тогда три варианта обхода этого дерева порождают три формы записи арифметического выражения:

1) КЛП − префиксную запись

− \* + *a b c* / *d* + *e* \* *f g* ;

2) ЛКП − инфиксную запись (без скобок, необходимых для задания последовательности выполнения операций)

*a* + *b* \* *c* − *d* / *e* + *f* \* *g* ;

3) ЛПК − постфиксную запись

*a b* + *c* \* *d e f g* \* + / − .

## Выполнение работы.

Для решения задачи был разработан графический интерфейс с помощью QtCreator. В интерфейсе реализовано поле для записи дерева в постфиксной форме и поле для вывода в инфиксной форме, кнопка начала обработки, а также поле графической интерпретации дерева.

Таблица 1 – Слоты класса MainWindow и их назначение

|  |  |
| --- | --- |
| Метод | Назначение |
| void on\_openFile\_clicked() | Считывание из файла |
| void on\_saveFile\_clicked() | Запись из поля вывода в файл |
| void on\_start\_clicked() | Запуск алгоритма |

Для реализации бинарного дерева были созданы структуры узла Node и самого дерева BinTree.

Также были реализованы функции, создающие и изменяющие бинарное дерево, некоторые из них приведены в табл. 2.

Таблица 2 – Основные функции работы с бинарным деревом

|  |  |
| --- | --- |
| Функция | Назначение |
| BinTree \*createBinTree() | Создает пустое бинарное дерево |
| BinNode \*createBinNode (char info, bool isNum) | Создает узел с заданными полями |
| BinNode \*appendRight(Node \*node, char info, bool isNum) | Создает узел с заданными полями, устанавливая его как правый к заданному узлу |
| BinNode \*appendLeft(Node \*node, char info, bool isNum) | Создает узел с заданными полями, устанавливая его как левый к заданному узлу |
| BinNode \*setInfo(Node \*node, char info, bool isNum) | Изменяет поля в заданном узле |
| int countDeep(Node \*&node) | Возвращает количество уровней дерева |
| BinTree \*create (QStringList in, int &err) | Создает бинарное дерево из массива строк-элементов, полученного из входной строки |
| QString getInfixNotation (Node \*root) | Возвращает строку, представляющую дерево в инфиксной форме |
| QString modding(BinNode \*root) | Преобразовывает дерево |

Программа имеет возможность графического отображения полученного бинарного дерева с помощью виджета QGraphicsView. Функции, необходимые для связи графического интерфейса и алгоритмов обработки дерева, а также для графического представления дерева, представлены в табл. 3.

Таблица 3 – Функции, связующие графический интерфейс и алгоритмы

|  |  |
| --- | --- |
| Функция | Назначение |
| QString processing (QString input, BinTree \*&tree) | Выполняет считывание и обработку данных, возвращая результат в виде строки |
| QGraphicsScene \*graphic(BinTree \*tree, QGraphicsScene \*&scene) | По заданному бинарному дереву выполняет рисование в объекте QGraphicsScene |
| int paint(QGraphicsScene \*&scene, BinNode \*node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth) | Рекурсивный алгоритм обхода дерева и рисования узлов в заданном объекте QGraphicsScene |
| int beginning(QTextEdit \*&uiInput, QTextEdit \*&uiOutput, QGraphicsScene \*&scene) | Перенос данных из графического интерфейса в функции обработки для обработки и вывода |

## Оценка эффективности алгоритма.

Алгоритм создания бинарного дерева по строке является итеративным, каждый элемент строки обрабатывается один раз, а значит сложность алгоритма можно оценить как .

Алгоритм вывода дерева в инфиксной форме, либо же в виде уступчатого списка является рекурсивным, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма также .

Алгоритм изменения дерева по формулам является рекурсивным и на каждом шаге в случае подозрения на соответствие формулы требует обхода поддеревьев текущего листа. Обход поддеревьев имеет сложность , глубина рекурсии также имеет сложность , что приводит к сложности алгоритма .

## Тестирование программы.

|  |  |  |  |
| --- | --- | --- | --- |
| Набор тестовых данных | | Предполагаемые результаты, высчитанные вручную | Результаты выполнения программы |
| № | Данные |
| 1 | ( ( ( 9 2 - ) ( 3 4 + ) \* ) ( 3 4 - ) + ) | ((( 9 - 2 ) \* ( 3 + 4 )) + ( 3 - 4 )) | ((( 9 - 2 ) \* ( 3 + 4 )) + ( 3 - 4 )) |
| 2 | ( ( ( 0 2 + ) ( 3 0 + ) \* ) ( 3 4 - ) + ) | (( 2 \* 3 ) + ( 3 - 4 )) | (( 2 \* 3 ) + ( 3 - 4 )) |
| 3 | ( ( ( 7 0 \* ) ( 3 0 - ) + ) ( 3 4 - ) + ) | ( 3 + ( 3 - 4 )) | ( 3 + ( 3 - 4 )) |
| 4 | ( ( ( 7 0 + ) ( 3 0 - ) + ) ( 1 0 - ) + ) | (( 7 + 3 ) + 1 ) | (( 7 + 3 ) + 1 ) |
| 5 | ( ( ( ( 5 1 / ) 2 + ) ( 3 5 + ) \* ) ( 3 4 - ) + ) | (((( 5 / 1 ) + 2 ) \* ( 3 + 5 )) + ( 3 - 4 )) | (((( 5 / 1 ) + 2 ) \* ( 3 + 5 )) + ( 3 - 4 )) |

## Выводы.

В ходе выполнения лабораторной работы была написана программа, создающая бинарное дерево согласно выражению в постфиксной форме, восстанавливающая выражение из дерева в инфиксной форме, изменяющая дерево согласно правилу вынесения общего множителя, представляющая дерево в виде уступчатого списка, а также графически.

# Приложение А Исходный код программы modify.cpp

#include "modify.h"

#include "graphic.h"

int **beginning**(QTextEdit \*&uiInput, QTextEdit \*&uiOutput, QGraphicsScene \*&scene)

{

BinTree \*tree = nullptr;

QString output = processing(uiInput->toPlainText(), *tree*);

uiOutput->setPlainText(output);

if (!output.count("Error"))

graphic(tree, *scene*);

else

scene->clear();

return 0;

}

QString **processing**(QString input, BinTree \*&tree)

{

QString output;

int err = 0;

QStringList inArr = input.split(" ");

tree = create(inArr, *err*);

switch (err)

{

case 3:

output += "Только циферки буковки скобочки\n";

return output;

case 4:

output += "Постфикс не оч\n";

return output;

}

output += "Инфиксная форма до преобразования:\n";

output += getInfixNotation(tree->root); // ЛКП

output += modding(tree->root); // замена формул

output += "\n\nИнфиксная форма после преобразования:\n";

output += getInfixNotation(tree->root); // ЛКП

return output;

}

QString **modding**(Node \*root)

{

QString output;

if (root == nullptr)

return "";

output += modding(root->left);

output += modding(root->right);

if (root->info == '+' && root->left && root->right)

{

if (root->right->info == '0' && isdigit(root->left->info))

{

root->info = root->left->info;

root->left = nullptr;

root->right = nullptr;

}

else if (root->left->info == '0' && isdigit(root->right->info))

{

root->info = root->right->info;

root->right = nullptr;

root->left = nullptr;

}

}

else if (root->info == '-' && root->left && root->right)

{

if (root->right->info == '0' && isdigit(root->left->info))

{

root->info = root->left->info;

root->left = nullptr;

root->right = nullptr;

}

}

else if (root->info == '\*' && root->left && root->right)

{

if (root->right->info == '1' && isdigit(root->left->info))

{

root->info = root->left->info;

root->left = nullptr;

root->right = nullptr;

}

else if (root->left->info == '1' && isdigit(root->right->info))

{

root->info = root->right->info;

root->right = nullptr;

root->left = nullptr;

}

else if (root->left->info == '0' && isdigit(root->right->info))

{

root->info = '0';

root->right = nullptr;

root->left = nullptr;

}

else if (root->right->info == '0' && isdigit(root->left->info))

{

root->info = '0';

root->right = nullptr;

root->left = nullptr;

}

}

return output;

}

QString **getInfixNotation**(Node \*root)

{

if (root == nullptr)

return "";

QString output;

if (root->left || root->right)

output += "(";

output += getInfixNotation(root->left);

output += " ";

output += root->info;

output += " ";

output += getInfixNotation(root->right);

if (root->left || root->right)

output += ")";

return output;

}

# Приложение Б Исходный код программы modify.h

#ifndef MODIFY\_H

#define MODIFY\_H

#include "bt.h"

int **beginning**(QTextEdit \*&uiInput, QTextEdit \*&uiOutput, QGraphicsScene \*&scene);

QString **processing**(QString input, BinTree \*&tree);

QString **modding**(Node \*root);

#endif // MODIFY\_H

# Приложение b Исходный код программы mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

scene = new QGraphicsScene;

ui->graphicsView->setScene(scene);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_start\_clicked**()

{

beginning(*ui->input*, *ui->output*, *scene*);

}

void MainWindow::**on\_openFile\_clicked**()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("load"), QDir::homePath(), tr("\*.txt"));

if (QString::compare(fileName, QString()) != 0)

{

ifstream f(qPrintable(fileName), ios::in);

string out;

getline(f, *out*);

f.close();

ui->input->setPlainText(QString::fromStdString(out));

}

}

void MainWindow::**on\_saveFile\_clicked**()

{

QString filePath = QFileDialog::getSaveFileName(this, tr("save"), QDir::homePath(), tr("\*.txt"));

if (QString::compare(filePath, QString()) != 0)

{

ofstream ff(qPrintable(filePath));

ff << qPrintable(ui->output->toPlainText());

ff.close();

}

}

# Приложение г Исходный код программы mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "modify.h"

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **on\_start\_clicked**();

void **on\_saveFile\_clicked**();

void **on\_openFile\_clicked**();

private:

Ui::MainWindow \*ui;

QGraphicsScene \*scene;

};

#endif // MAINWINDOW\_H

# Приложение д Исходный код программы main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

# Приложение е Исходный код программы graphic.cpp

#include "graphic.h"

QGraphicsScene \***graphic**(BinTree \*tree, QGraphicsScene \*&scene)

{

if (tree == nullptr)

return scene;

scene->clear();

QPen pen;

pen.setWidth(5);

QColor color;

color.setRgb(192, 192, 192);

pen.setColor(color);

QBrush brush (color);

QFont font;

font.setFamily("Helvetica");

int w\_deep = static\_cast<int>(pow(2, tree->deep)+2);

int h = 60;

int w = 12;

font.setPointSize(w);

int width = (w\*w\_deep)/2;

paint(*scene*, tree->root, width/2, h, w, h, *pen*, *brush*, *font*, w\_deep);

return scene;

}

int **paint**(QGraphicsScene \*&scene, Node \*node, int width, int height, int w, int h, QPen &pen, QBrush &brush, QFont &font, int depth)

{

if (node == nullptr)

return 0;

QString out;

out += node->info;

QGraphicsTextItem \*elem = new QGraphicsTextItem;

elem->setPos(width, height);

elem->setPlainText(out);

elem->setFont(font);

scene->addRect(width-w/2, height, w\*5/2, w\*5/2, pen, brush);

if (node->left != nullptr)

scene->addLine(width+w/2, height+w, width-(depth/2)\*w+w/2, height+h+w, pen);

if (node->right != nullptr)

scene->addLine(width+w/2, height+w, width+(depth/2)\*w+w/2, height+h+w, pen);

scene->addItem(elem);

paint(*scene*, node->left, width-(depth/2)\*w, height+h, w, h, *pen*, *brush*, *font*, depth/2);

paint(*scene*, node->right, width+(depth/2)\*w, height+h, w, h, *pen*, *brush*, *font*, depth/2);

return 0;

}

# Приложение Ж Исходный код программы graphic.h

#ifndef GRAPHIC\_H

#define GRAPHIC\_H

#include "modify.h"

QGraphicsScene \***graphic**(BinTree \*tree, QGraphicsScene \*&scene);

int **paint**(QGraphicsScene \*&scene, Node \*node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);

#endif // GRAPHIC\_H

# Приложение З Исходный код программы bt.cpp

#include "bt.h"

BinTree \***createBinTree**()

{

BinTree \*tree = new BinTree;

tree->root = nullptr;

tree->deep = 0;

return tree;

}

Node \***createBinNode**(char info, bool isNum)

{

Node \*node = new Node;

node->info = info;

node->isLeaf = isNum;

node->left = nullptr;

node->right = nullptr;

return node;

}

Node \***appendRight**(Node \*node, char info, bool isNum)

{

node->right = createBinNode(info, isNum);

return node->right;

}

Node \***appendLeft**(Node \*node, char info, bool isNum)

{

node->left = createBinNode(info, isNum);

return node->left;

}

Node \***setInfo**(Node \*node, char info, bool isNum)

{

if (node == nullptr)

return nullptr;

node->info = info;

node->isLeaf = isNum;

return node;

}

int **countDeep**(Node \*&node)

{

if (node == nullptr)

return 0;

int cl = countDeep(*node->left*);

int cr = countDeep(*node->right*);

return 1 + ((cl>cr)?cl:cr);

}

int **updateDeep**(BinTree \*tree)

{

tree->deep = countDeep(*tree->root*);

return tree->deep;

}

BinTree \***create**(QStringList in, int &err)

{

stack <Node \*> BNStack;

BinTree \*tree = createBinTree();

tree->root = createBinNode('\0', 1);

Node \*temp = tree->root;

BNStack.push(temp);

for (int i=0; i<in.length(); i++)

{

if (in[i] == "(")

{

BNStack.push(temp);

temp = appendLeft(temp, '\0', 1);

}

else if (in[i] == ")")

{

if (BNStack.empty() || temp->info == '\0')

{

err = 4;

return tree;

}

if (temp == BNStack.top()->left)

{

temp = BNStack.top();

temp = appendRight(temp, '\0', 1);

}

else

{

temp = BNStack.top();

BNStack.pop();

}

}

else if (in[i] == "\*" || in[i] == "/" || in[i] == "-" || in[i] == "+")

setInfo(temp, qPrintable(in[i])[0], 0);

else

{

if (in[i].length() > 1 || ((in[i][0] < 'a' || in[i][0] > 'z') && (!in[i][0].isDigit())))

{

err = 3;

return tree;

}

if (BNStack.empty())

{

err = 4;

return tree;

}

setInfo(temp, qPrintable(in[i])[0], 1);

if (temp == BNStack.top()->left)

{

temp = BNStack.top();

temp = appendRight(temp, '\0', 1);

}

else

{

temp = BNStack.top();

BNStack.pop();

}

}

}

if (!BNStack.empty())

{

err = 4;

return tree;

}

updateDeep(tree);

return tree;

}

# Приложение И Исходный код программы bt.h

#ifndef BINTREE\_H

#define BINTREE\_H

#include <iostream>

#include <fstream>

#include <sstream>

#include <QMainWindow>

#include <QGraphicsItem>

#include <QGraphicsView>

#include <QGraphicsEffect>

#include <QFileDialog>

#include <QStandardPaths>

#include <QtGui>

#include <QLabel>

#include <QColorDialog>

#include <QInputDialog>

#include <QMainWindow>

#include <QPushButton>

#include <QMessageBox>

#include <QStringList>

#include <QTextEdit>

#include <stack>

using namespace std;

struct **Node**

{

bool isLeaf;

char info;

Node \*left, \*right;

};

struct **BinTree**

{

Node \*root;

int deep;

};

BinTree \***createBinTree**();

Node \***createBinNode**(char info, bool isNum);

Node \***appendRight**(Node \*node, char info, bool isNum);

Node \***appendLeft**(Node \*node, char info, bool isNum);

Node \***setInfo**(Node \*node, char info, bool isNum);

BinTree \***create**(QStringList in, int &err);

QString **getInfixNotation**(Node \*root);

int **countDeep**(Node \*&node);

int **updateDeep**(BinTree \*tree);

#endif // BINTREE\_H